Using `smartadata`

Ignacio Cordón

2019-12-18

Purpose of `smartdata`

R programming language has a wide variety of packages that target machine learning topiscs. However, each package has its own API, which makes the task of using several of them quite time-consuming. The main purpose of `smartdata` is to provide a common interface for a collection of well-used machine learning packages, so that using methods from different libraries gets easier. Also, it standardizes names of arguments, so that if a method had a parameter `num_iterations` and other had a parameter `iterations` with similar meaning, then both methods would take the same parameter name in `smartdata`.

`smartdata` includes preprocessing algorithms for oversampling, instance selection, feature selection, normalization, discretization, space transformation, outliers treatment, missing values imputation and noise cleaning.

Each of the aforementioned topics has its corresponding wrapper: `instance_selection`, `feature_selection`, `normalize`, `discretize`, `space_transformation`, `clean_outliers`, `impute_missing` and `clean_noise`, respectively.

In addition to that, `magrittr` has been used to provide a pipeline, so that a typical preprocessing workflow can be expressed as:

```
result <- dataset %>% impute_missing %>% clean_noise %>% oversample %>% feature_selection
```

Basic help

To check the methods a certain wrapper can be called with, we can simply call the `help` function included in `smartdata`: `which_options`.

```
library("smartdata")
which_options("instance_selection")
#> Possible methods are: 'CNN', 'ENN', 'multiedit', 'FRIS'
```

To check parameters for a wrapper, it suffices to do `?{name of the wrapper}`. For example: `?instance_selection`, which would output:

```
Usage

instance_selection(dataset, method, class_attr = "Class", ...)

Arguments

dataset     we want to perform an instance selection on
method      selected method of instance selection
class_attr  character. Indicates the class attribute from dataset. Must exist in it
...         Further arguments for method
```

The most common arguments that can appear in a wrapper documentation are:

- `dataset`: the `data.frame` we want to apply the method on.
- `method`: the selected procedure (for example, for instance selection, it could be `"multiedit"`, `"CNN"`, `"ENN"` or `"FRIS"`).
- `class_attr`: the column name or names (a `character` vector) that represents the class attribute inside the dataset. By default, `"class"`
- `exclude`: a `character` vector indicating which attributes to ignore in the procedure. The columns will be striped from the dataset and joined to the modified dataset after the procedure. Normally, if the number of columns has not been modified, the order of all columns will be preserved and the dataset will have the same shape before and after the preprocessing. In techniques such as feature selection, the order of all the columns included in the resulting dataset (columns selected by the procedure and excluded columns) will preserve the original sorting (if the i-th attribute appeared before the j-th in the original dataset, if they are also present in the result, their order will be the same).

Moreover, if we already know which method we want to use but we do not recall its arguments, we can check its list of parameters with:

```r
which_options("instance_selection", "multiedit")
#> For more information do: ?class::multiedit
#> Parameters for multiedit are:
#>   * k: Number of neighbors used in KNN
#>       Default value: 1
#>   * num_folds: Number of partitions the train set is split in
#>               Default value: 3
#>   * null_passes: Number of null passes to use in the algorithm
#>                 Default value: 5
```

That option provides a brief description for each possible parameter, as well as a reference to the original function, in case the information for some parameter might not seem clear enough (although mapping between original function's arguments and `smartdata` wrapper is not exact).

In summary, a valid call for `multiedit` would be:

```r
super_iris <- iris %>% instance_selection("multiedit", k = 3, num_folds = 2,
                                          null_passes = 10, class_attr = "Species")
```

Or:

```r
super_iris <- iris %>% instance_selection("multiedit", k = 3, null_passes = 10,
                                          class_attr = "Species")
```

Or:

```r
super_iris <- iris %>% instance_selection("multiedit", k = 3,
                                          class_attr = "Species")
```

Or even:

```r
super_iris <- iris %>% instance_selection("multiedit", class_attr = "Species")
```

Techniques included in `smartdata`

Let $S = \{(x_i, y_i)\}_{i=1^m}$ be our set of data from this moment on. It holds $x_i \in \mathcal{X} \subset \mathbb{R}^n$ and $y_i \in \mathcal{Y}$ where $\mathcal{Y}$ is a finite set of labels. (x_i, y_i) is called an instance.

Oversampling

Given $\mathcal{Y}$ with $|\mathcal{Y}| = 2$, if there exist more instances labeled with one class than with the other, oversampling will generate synthetic instances labeled with the minority class, namely E, so that the resulting dataset $S \cup E$ has better characteristics for classification.

Since oversampling is going to replicate or generate artificial instances belonging to the minority class, a class attribute must be indicated with `class_attr` argument, as well as a `ratio` (a desired proportion of imbalance, when applicable), where this ratio is computed as:

$$\frac{\text{number of minority instances}}{\text{number of majority instances}}$$

Also, a `filtering` argument can be provided, indicating whether to perform a filtering of the generated instances using NEATER.

Possible methods are:

- `PDFOS`
- `RWO`
- `ADASYN`
- `ANSMOTE`
- `SMOTE`
- `MWMOTE`
- `BLSMOTE`
- `DBSMOTE`
- `SLMOTE`
- `RSLSMOTE`
- `RACOG`
- `wRACOG`

Only if picked method is wRACOG, an additional parameter `wrapper` with possible values `'KNN'` and `'C5.0'` can be provided. That parameter denotes the desired classificator to select instances.

As example:

```r
data(iris0, package = "imbalance")
super_iris <- oversample(iris0, method = "MWMOTE", class_attr = "Class",
                         ratio = 0.8, filtering = TRUE)
```

Instance selection

Consists in picking a subset $S' \subseteq S$, so that certain characteristics are preserved with respect to the original sample (such as distribution of classes) or at least we keep most of representative instances.

As said, methods try to pick instances preserving original classes distribution, so `class_attr` must be supplied.

Possible methods are:

- `CNN`
- `ENN`
- `multiedit`
- `FRIS`

As example:

```r
super_iris <- instance_selection(iris, method = "CNN", class_attr = "Species")
```

Feature selection

Consists in given a subset of features $T \subseteq \{1, \dots n\}$, projecting the tuples of $S = \{(x_i, y_i)\}_{i=1}^m$ to the set of features given by T. That is, if we defined

$$p_T((x_{i1}, \dots, x_{in})) = (x_{ij})_{j \in T}$$

as the projection to the features in T, doing a feature selection would result in a set $S' = \{(p_T(x_i), y_i)\}_{i=1}^m$, if the picked features were T.

Apart from the parameters for each specific method, `class_attr` must be supplied for each of the preprocessings, and an additional parameter `exclude` can be supplied with a vector of features names, so that those features are striped before the feature selection and joined after the procedure.

Possible methods are:

- `Boruta`
- `chi_squared`
- `information_gain`
- `gain_ratio`
- `sym_uncertainty`
- `oneR`
- `RF_importance`
- `best_first_search`
- `forward_search`
- `backward_search`
- `hill_climbing`
- `cfs`
- `consistency`

As example:

```r
super_iris <- feature_selection(iris, "Boruta", class_attr = "Species")
```

Normalization

Implies converting the sample data S into another dataset S' where each tuple is treated so that standard deviation of the data ends up being zero (feature-wise or considered as elements of $\mathbb{R}^n$), or all the data is mapped to $[0, 1]$ interval (e.g. dividing by the maximum feature-wise), etc

Possible methods are:

- `z_score`
- `pos_standardization`
- `unitization`
- `pos_unitization`
- `min_max`
- `rnorm`
- `rpnorm`
- `sd_quotient`
- `mad_quotient`
- `range_quotient`
- `max_quotient`
- `mean_quotient`
- `median_quotient`
- `sum_quotient`
- `ssq_quotient`
- `norm`

- pnorm
- znorm
- decimal_scaling
- sigmoidal
- softmax

Apart from the parameters for each specific method, an additional parameter `exclude` can be supplied with a vector of feature names, so that those features are striped before the normalization and joined after the procedure, with the unchanged original data.

As an example:

```
super_iris <- normalize(iris, method = "min_max", exclude = "Species", by = "column")
```

Discretization

Discretizing a dataset consists in turning continuous variables into discrete attributes, that is, picking an attribute which can take real values and making a binning so that there exists a finite set of values for the variable. This procedure can have a great importance in procedures such as classification.

Possible methods are:

- chi2
- chi_merge
- extended_chi2
- mod_chi2
- CAIM
- CACC
- ameva
- mdlp
- equalfreq
- equalwidth
- globalequalwidth

Similar to feature selection or normalization wrappers, `discretize` needs a `class_attr` in certain cases (methods like `equalfreq`, `equalwidth` or `globalequalwidth` do not need it, and will output an error if this argument is supplied to `discretize`).

Also an `exclude` argument can be passed to the method, indicating which variables are going to be ignored when performing the process.

As an example:

```
super_iris <- discretize(iris, method = "chi2", class_attr = "Species")
```

Space transformation

Space transformation changes a training set $S = \{(x_i^{(1)}, \dots x_i^{(n)})\}_{i=1\dots m}$ where $x_i^{(j)} \in \mathbb{R}$ into another set $S' = \{(\bar{x}_i^{(1)}, \dots, \bar{x}_i^{(k)})\}_{i=1\dots m}$ with better properties for machine learning methods.

Possible methods are:

- lle_knn
- lle_epsilon
- adaptative_gpca

This treatment accepts an `exclude` argument in form of a vector of column names. Those columns will be ignored during the procedure and appended in the same order without variation after it, before the non numeric columns (which will be appended in the precise sequence they appeared originally).

As example:

```r
data(ecoli1, package = "imbalance")
super_ecoli <- space_transformation(ecoli1, "lle_knn", k = 3, num_features = 2,
                                    regularization = 1, exclude = c("Mcg", "Alm1"))
```

Outliers

Outliers are instances that lie far from the others (that is, they seem quite distant from the others, using some kind of predefined distance or measure).

Possible methods are:

- `multivariate`
- `univariate`

When treating outliers, those can be considered as whole rows of the dataset (that is, an outlier is a whole instance), or as certain values inside each column (that is, outliers inside attributes). The first case receives the name of `multivariate` approach, whereas the second case is called `univariate`.

`multivariate` approach will imply deletion of the outlier instances from the dataset. `univariate` outliers can be treated imputing the median or the mean of the column.

As example:

```r
super_iris <- clean_outliers(iris, method = "multivariate", type = "adj")
super_iris <- clean_outliers(iris, method = "univariate", type = "z", fill = "mean")
```

Missing values

Missing values are tuples that lack a certain attribute value, so a preliminary imputation could have a lot of benefits with respect to a posterior machine learning technique, such as classification.

Possible methods are:

- `gibbs_sampling`
- `expect_maximization`
- `central_imputation`
- `knn_imputation`
- `rf_imputation`
- `PCA_imputation`
- `MCA_imputation`
- `FAMD_imputation`
- `hotdeck`
- `iterative_robust`
- `regression_imputation`
- `ATN`

An `exclude` argument in form of a vector of column names can be passed to `impute_missing`, so that those columns will be ignored during the procedure and appended without variation after it.

As example:

```r
data(nhanes, package = "mice")
super_nhanes <- impute_missing(nhanes, "gibbs_sampling")
```

Noise

Noisy data are instances which include additional meaningless information apart from the true information they encode. Removing or repairing those instances prior to a classification can have a potential benefit on it.

Possible methods are:

- `AENN`
- `ENN`
- `BBNR`
- `DROP1`
- `DROP2`
- `DROP3`
- `EF`
- `ENG`
- `HARF`
- `GE`
- `INFFC`
- `IPF`
- `Mode`
- `PF`
- `PRISM`
- `RNN`
- `ORBoost`
- `edgeBoost`
- `edgeWeight`
- `TomekLinks`
- `dynamic`
- `hybrid`
- `saturation`
- `consensusSF`
- `classificationSF`
- `C45robust`
- `C45voting`
- `C45iteratedVoting`
- `CVCF`

A `class_attr` must be supplied.

As example:

```r
super_iris <- clean_noise(iris, method = "AENN", class_attr = "Species", k = 3)
```